Localisation, Recognition and Expression of Ornaments in Music Scores

Bonnici, Alexandra; Abela, Julian; Zammit, Nicholas; Azzopardi, George

Published in:
DocEng '18 Proceedings of the ACM Symposium on Document Engineering 2018

DOI:
10.1145/3209280.3209536

Automatic Ornament Localisation, Recognition and Expression from Music Sheets

Alexandra Bonnici
Faculty of Engineering
University of Malta, Malta
alexandra.bonnici@um.edu.mt

Julian Abela
Faculty of ICT
University of Malta, Malta
julian.abela.14@um.edu.mt

Nicholas Zammit
Faculty of Engineering
University of Malta, Malta
nicholas.zammit.14@um.edu.mt

George Azzopardi
Faculty of Science and Engineering
University of Groningen, the Netherlands
g.azzopardi@rug.nl

ABSTRACT

Musical notation is a means of passing on performance instructions with fidelity to others. Composers, however, often introduced embellishments to the music they performed notating these embellishments with symbols next to the relevant notes. In time, these symbols, known as ornaments, and their interpretation became standardized such that there are acceptable ways of interpreting an ornament. Although music books may contain footnotes which express the ornament in full notation, these remain cumbersome to read. Ideally, a music student will have the possibility of selecting ornamented notes and express them as full notation. The student should also have the possibility to collapse the expressed ornament back to its symbolic representation, giving the student the possibility of also becoming familiar with playing from the ornamented score. In this paper, we propose a complete pipeline that achieves this goal. We compare the use of cosfire and template matching for optical music recognition to identify and extract musical content from the score. We then express the score using MusicXML and design a simple user interface which allows the user to select ornamented notes, view their expressed notation and decide whether they want to retain the expressed notation, modify it, or revert to the symbolic representation of the ornament. The performance results that we achieve indicate the effectiveness of our proposed approach.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Applied computing** → **Sound and music computing**; **Document preparation**;

ACM Reference Format:
Alexandra Bonnici, Julian Abela, Nicholas Zammit, and George Azzopardi. 2018. Automatic Ornament Localisation, Recognition and Expression from Music Sheets. In *DocEng '18: ACM Symposium on Document Engineering 2018, August 28–31, 2018, Halifax, NS, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3209280.3209536

DocEng '18, August 28–31, 2018, Halifax, NS, Canada

ACM ISBN 978-1-4503-5769-2/18/08…$15.00
https://doi.org/10.1145/3209280.3209536

1 INTRODUCTION

Musical notation is a means through which a composer expresses the way that a composition should be performed and is a means of passing on performance instructions with fidelity to others. Composers, however, often introduced embellishments to the music they performed. Rather than notating these embellishments in full, composers typically notated these embellishments with symbols next to the relevant notes. In time, these symbols, known as ornaments, and their interpretation became standardised such that there are acceptable ways of interpreting an ornament, although, the interpretation may vary with the note duration, the tempo of the music, the notes preceding the embellished note as well as the skill of the performer.

We can, therefore, think of ornaments as a neat, short-hand way of condensing the embellished notes for quick reading. However, for inexperienced music learners, this may come at a cost. Properly executing an embellished note requires knowing what notes to play, the proper rhythm for these notes, and in the case of piano players, coordinating these additional notes with the rhythms played with the other hand. While all of these aspects become second nature to experienced musicians, the proper execution of ornaments can be a stumbling block for a music student. Editors of books designed for tuition are well aware of the interpretation problems that aspiring musicians may face and provide extra notes illustrating the proper execution of the ornaments. At best, these notes are presented as a full five-line stave above the system, at worst, as footnotes as shown in Figure 1. In either case, the editor provides a single explanation of the ornament, at its first occurrence and the student will need to refer to this for all other occurrences of the same ornament, shifting the notes to other starting points as necessary. In the particular example shown in Figure 1, there are six different interpretations of the turn ornament which are brought about by the different rhythmic qualities of the notes on which the turn ornament is applied. This excerpt has an additional four turn ornaments which the editor leaves unmarked and whose interpretation is obtained by comparing with one of the previously marked turn ornaments. The variability in the interpretation of any single ornament, coupled with the need to simultaneously fit the ornament interpretation with other notes exhibiting

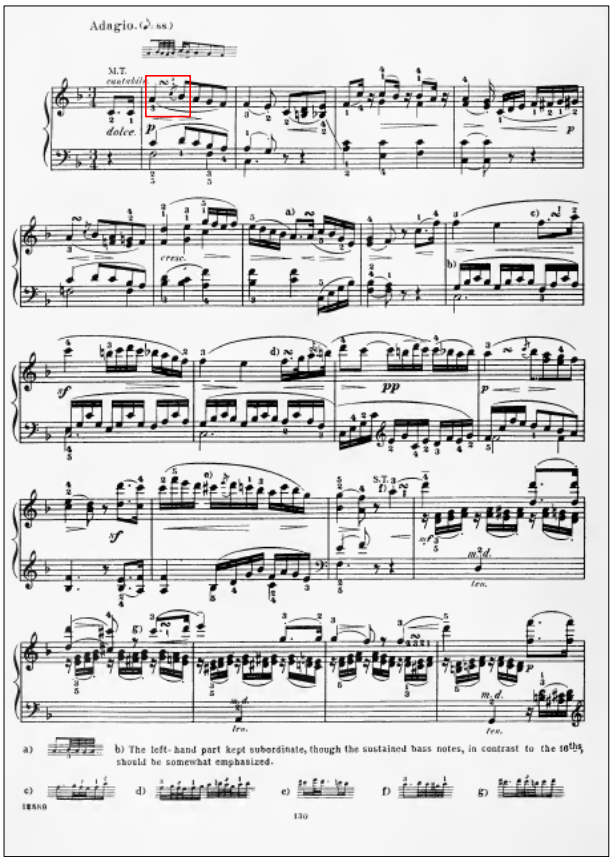


Figure 1: An excerpt from Beethoven's Piano Sonata Op. 2, No. 1, 2nd Movement. In this example, the ornament on the first system is expressed on top of the system, while six footnotes express the ornaments on the remaining systems. The MusicXML representation of the notes enclosed in the box is listed in Listing 2

different rhythmic patterns makes the proper execution of ornaments a daunting task for students and amateur musicians [17].

A solution to this problem would be to re-write the music, expressing all ornaments in full and practising from the new score until the learner masters all ornaments in the piece. However, re-writing pages of music can be time-consuming. Occasionally, through music depositories, it is possible to find musical scores notated for digital readers such as MuseScore[1] among others, however, while this can interpret some ornaments, it does not provide the full, written notation of the ornaments. Nor is there an easy way to adjust the in-built interpretation of that ornament to suit tempo or student ability. Applications which can perform optical music recognition (OMR), such as SharpEye[2] exist, but while this has some support for the recognition of ornaments, it does not offer the possibility of expressing the ornaments.

Ideally, a music student has the possibility of selecting ornamented notes and expresses them as full notation. The student should also have the possibility to collapse the expressed ornaments back to their symbolic representations, giving the student the possibility to become familiar with playing from the ornamented representation once the execution is mastered. Switching between the two modalities should be quick and effortless, and as different ornaments require different levels of skill, it should also be possible to create intermediary score representations which contain a mixture of written out and symbolic representations of the ornaments.

In this paper, we document our efforts in creating such a system. We describe the use of COSFIRE for optical music recognition, taking particular note of the recognition of ornament symbols and comparing the performance of the proposed COSFIRE approach with the use of template matching techniques described in the literature. We then express the musical content extracted from the score using MusicXML and design a simple user interface which allows the user to select ornamented notes, view their expressed notation and decide whether they want to retain the expressed notation, modify it, or revert to the symbolic representation of the ornament. For the purpose of this work, we focus on score images obtained from recent publications of music books, such as modern anthologies, tuition books or examination pieces.

The rest of this paper is organised as follows: Section 2 discusses the related works described in the literature, Section 3 presents our proposed optical music recognition algorithms, Section 4 describes our approach to writing the MusicXML file, while Section 5 describes how we express ornaments in full and our user interface. Section 6 describes the evaluation protocol adopted in this work, with results presented in Section 7. Finally, we draw conclusions in Section 8.

2 RELATED WORK

Optical music recognition (OMR) systems consist of three main steps, namely image pre-processing, symbol recognition and musical reconstruction [15]. The role of the image pre-processing step is to simplify the image, adjusting it so that subsequent symbol recognition may be more robust. It typically includes standard pre-processing algorithms such as noise filtering and binarisation [10], which are common to document image analysis but may also extend to music-specific pre-processing such staff-line removal. This is the process which frees the note and other symbols from the underlying staff lines such that the symbol recognition is performed on images containing only symbols [23]. There are a variety of techniques for staff-line removal described in the literature including the use of run-lengths [9], wavelets [5], horizontal projections [2], morphology [23], path following [18] and the Hough Transform [6], among others. A commonality among these algorithms is the evaluation of the thickness of each line forming the staff as well as the thickness of the space between the lines. These two thickness values provide a measure of the scale of the image and hence the size of the note symbols [15]. Although staff-line removal is intended to reduce the burden of the symbol recognition step, the removal of staff lines may fragment the symbols if the performance of the staff line removal is not adequate [2]. For this reason, although the majority of the OMR applications perform staff-line segmentation, this is not necessarily always the case. Indeed Tambouratzis

[1] https://musescore.com/
[2] http://www.visiv.co.uk/

[19] adopts the staff lines in the symbol prototypes and performs the symbol recognition directly on the image without first removing staff lines. Others, such as Rossant and Bloch [16] and Toyama et al. [21] perform symbol segmentation through correlation and do not require separate staff line removal, while Lee et al. [12] and Nguyen and Lee [14] who use vertical projects for symbol segmentation, acknowledge the contribution of the staff lines as a baseline in the vertical projections, but do not remove them.

The pre-processing steps are followed by symbol recognition algorithms which may be applied to the music score using pixel level information or to some higher level function that describes the symbol [15]. At a pixel level, symbol recognition may be applied to primitives such as line segments [21], glyphs that form parts to the musical symbol [13] or the symbol in its entirety [19]. Algorithms such as that described in Lee et al. [12], for example, look beyond the pixel level and represent the symbols by their vertical projections, performing the symbol classification on the projections. Others, such as Baró et al. [3], use a multi-modal approach, whereby the user is asked to create a higher level representation of the score by tracing over the score on a digital device with the symbol recognition being performed on both versions of the score.

Besides differences in the symbols or primitives selected, the different algorithms described in the literature use different techniques to perform symbol recognition. There are algorithms which use simple pattern recognition techniques such as template matching [2, 21], morphology operations [13] and Hough transform [2]. Other algorithms based on probabilistic approaches [19], convolutional neural networks [11], recurrent neural networks [3] and support vector machines [14] are also described in the literature.

These approaches perform symbol recognition on isolated symbols, whether in part or as a whole. Music symbols, however, should be interpreted in groups rather than as individuals and thus, a common feature in OMR algorithms is to organise the symbols in such a way as to obtain musical meaning from the symbols [2]. Such post-processing of the detected symbols may be based on heuristics derived from domain knowledge which describe the relative positions of symbols to each other [21]. These heuristics may be applied definite clause grammars [2, 8], as notation graphs [11] or through the use of fuzzy modelling [16]. Alternatively, van der Wel and Ullrich [22] perform symbol recognition of full lines of sheet music rather than individual symbols, using a sequence-to-sequence architecture and casting the problem into a translation problem. Such an approach resolves issues with fragmented symbols but is limited to monophonic music.

The result of OMR applications is an alternative representation of the musical score, traditionally using the Music Instrument Digital Interface (MIDI) file format, the Notation Interchange File Format (NIFF) and more recently, the MusicXML file format. The scope of the OMR has always been to represent the digital score faithfully and little has been done to extend the interpretation beyond the written score. Algorithms such as that described in [7] perform some degree of interpretation, using a graph representation of the music and casting the OMR problem into an optimisation problem to allow for transposition. However, to our knowledge, OMR systems seldom recognise ornaments and those which do, do not provide any interpretation of the ornament.

3 OPTICAL MUSIC RECOGNITION

The optical music recognition approach we adopt in this work consists of a pre-processing step to remove staff lines and segment the score into systems and bars. After pre-processing the score, we perform symbol recognition step to locate and classify all symbols in the score. Finally, we perform a score interpretation step which assigns a musical meaning to the detected symbols, allowing us to represent the pictorial score as a MusicXML file. Figure 2 illustrates the proposed pipeline of the optical music recognition steps used in this application. The following sections describe the steps involved.

3.1 Score Pre-processing

We start the score image pre-processing by performing image binarisation to reduce the grey-scale image into a binary image. In this application, we use the Otsu algorithm to automatically determine a threshold suitable for each score image. After binarisation, we perform skew correction, using the algorithm described in [9], following which we proceed to separate the staff lines from the notes, and other performance symbols to simplify the image for subsequent symbol recognition. In this application, we use run-length encoding to classify the pixels as being either staff lines or symbols.

In binary images, run-length encoding records the number of consecutive black or white pixels along a specified direction. Thus, vertical and horizontal run-length encoding capture different information about the pattern formed by the underlying staff lines that form the music score.

In vertical run-length encoding, the most frequently occurring black run-lengths correspond to the height h_L of the staff lines. Similarly, the most frequently occurring white run-lengths correspond to the separation h_S between the staff lines. We alter the run-lengths by converting each black run whose length is longer than h_L into a white run of the same length. In this manner, when decoding the run-lengths back into an image, we obtain an image L_v consisting of only staff lines. To obtain an image consisting of only symbols, we compute the intersection $S_v = I \cap \bar{L}_v$, where I is the binary score image. In practice, however, there may be parts of symbols with vertical run-lengths equal to h_L, resulting in misclassified pixels and hence, fragmented symbols in S_v as shown in Figure 3(b).

Applying horizontal run-length encoding to the score image results in long black runs which correspond to the length of the staff lines. Notes and symbols with white interior regions, such as flats or semibreves will, however, break these long runs into shorter black-runs. Nevertheless, run-lengths corresponding to staff lines remain longer than those obtained from other symbols. Thus, we alter the horizontal run-lengths by changing any black run shorter than a threshold t_h to a white run of the same length so that in decoding the horizontal run-lengths, we obtain a second staff line image L_h. The intersection $S_h = I \cap \bar{L}_h$ again isolates the symbols from the staff lines. Since the score image consists of symbols superimposed on staff lines, a black pixel may simultaneously be a staff line and a symbol. In horizontal run lengths, however, any such pixel contributes to the runs which we label as staff lines. Thus, the staff line image L_h contains pixels which could also be

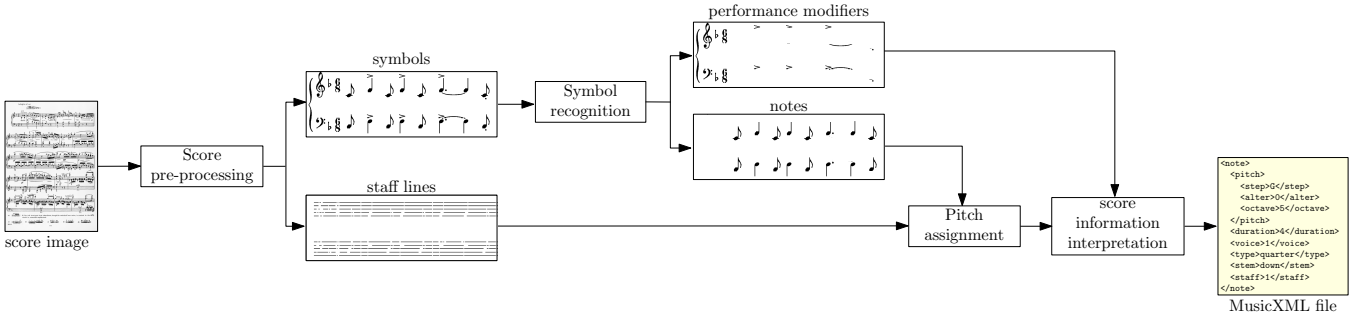


Figure 2: The proposed optical music recognition pipeline

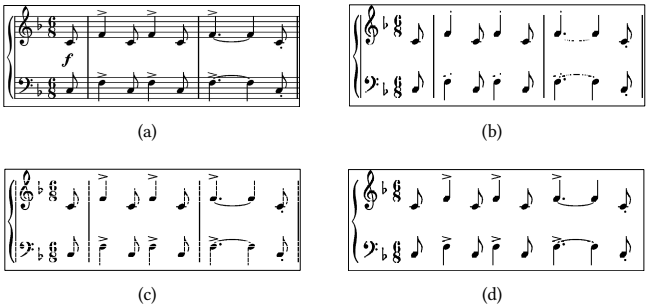


Figure 3: Staff line removal applied to (a) using vertical run-lengths (b) horizontal run-lengths and (c) the combination of vertical and horizontal run-lengths (d)

part of symbols. As a result, S_h may also contain fragmented symbols as shown in Figure 3(c). We note, however, that the misclassified pixels in S_h and S_v do not overlap such that we can obtain the full set of symbols through the union $S = S_h \cup S_v$ as shown in Figure 3(d). Likewise, we obtain the staff lines from the intersection $L = L_h \cap L_v$.

For the final step in our score pre-processing, we separate the music score into systems and each system into bars. A system consists of two or more staff lines connected by at least one bar-line at the beginning of each system. Each system is, therefore, a single connected component, distinct from other systems in the score and this allows us to segment the score into single systems [4]. We then apply the Hough transform to locate the vertical lines in each system. Here, the estimate of the staff height which we obtain from the run-length encoding allows us to distinguish between note stems and bar-lines or repeat-line symbols as detailed in [4]. By identifying the bar-lines, we can associate each symbol mark in S with the bar number to which it belongs.

3.2 Symbol recognition

In this work, we compare two symbol localisation and recognition approaches, namely a template matching approach used in [21] and the combination of shifted filter responses (COSFIRE) approach [1]. Both approaches require the use of templates or prototypes. Since we are using printed scores as the input source, we may obtain the required templates from the glyphs that define the music fonts used in music engraving software. In this work, we obtain templates from the Emmentaler font set used in engravers such as LilyPond[3].

Symbol recognition typically requires robustness to variations in scale, orientation and reflections of the symbol from the template glyph used to train the recogniser. Generally, increasing the robustness of the symbol recogniser to such variations will also incur an increase in the computational costs of the recongiser. Thus, it is sensible to limit, if possible, the degree of scale, orientation and reflection invariance. Assuming that the score image being processed is an upright image of the score, then the expected scale of the musical symbols may be deduced from the staff height. Since musical symbols do not generally experience drastic changes in scale within the score, the size of the height of the staff places a natural upper and lower limit on the degree of scale invariance required. Likewise, rotation invariance can be limited to 90-degree rotations of a subset of the musical symbols such as stem-notes and the staccatissimo symbols.

3.2.1 Template matching approach. Template matching involves computing the cross-correlation between the template of a symbol and the image, identifying a match if the cross-correlation value exceeds some threshold [21]. Cross-correlation is dependent on the size of the template and since symbols have different sizes, selecting a single threshold for all symbols is not possible. Thus, the normalised cross-correlation is used. This is defined as:

$$\gamma(u,v) = \frac{\sum_{x,y}(I(x,y) - \bar{I}_{u,v})(t(x-u,y-v) - \bar{t})}{\sqrt{\sum_{x,y}(I(x,y) - \bar{I}_{u,v})^2 (t(x-u,y-v) - \bar{t})^2}} \quad (1)$$

where $\bar{t}$ is the mean of the template image and $\bar{I}_{u,v}$ is the mean of the local pattern in the image under the template. The cross-correlation value is normalised to the range $[-1, 1]$ independent of the template size. To obtain the required scale, rotation and reflection invariance to detect all occurrences of all symbols, we provide different templates for each anticipated variance.

3.2.2 COSFIRE approach. Unlike template matching, the COSFIRE algorithm does not use the template image directly but configures keypoints which describe the key features of a given prototype around a central point, referred to as the COSFIRE support centre [1]. In the configuration stage, the COSFIRE algorithm applies

[3] http://lilypond.org/

a bank of orientation-selective Gabor filters and extracts information about the local maximum Gabor responses along a set of concentric circles with given radii ρ. The algorithm selects the polar coordinates (ρ_i, ϕ_i) of every keypoint i along with the scale λ_i and orientation θ_i parameters of the Gabor filter that achieves the maximum response at that position. The algorithm, therefore, describes the keypoint i by the tuple $(\lambda_i, \theta_i, \rho_i, \phi_i)$. Each prototype or template is described by a COSFIRE filter defined as a set of keypoints: $C = \{(\lambda_i, \theta_i, \rho_i, \phi_i)|i = 1, \cdots, n\}$ where n is the total number of keypoints detected in the given template.

A COSFIRE filter is applied is as follows. For each tuple i in the set C, we apply a Gabor filter with the scale λ_i and orientation θ_i. Then, in order to allow for some tolerance, we blur the Gabor responses with a max weighted pooling function. The weighting is achieved by a Gaussian function whose standard deviation σ_i grows linearly with the distance ρ_i from the support center of the COSFIRE filter: $\sigma_i = \sigma_0 + \alpha\rho_i$. For convenience reasons, the blurred Gabor responses are shifted by ρ_i pixels in the direction opposite to ϕ_i, so that the responses of all keypoints meet at the same position. Finally, the blurred and shifted Gabor responses are combined by the geometric mean. The local maximum responses in a COSFIRE filter output map indicate the locations at which local patterns, which are similar to the prototype used to configure the filter, are located. For further details we refer the reader to [1] which includes elaborate explanation on how a COSFIRE filter can achieve invariance to rotation, scale and reflection.

To apply the COSFIRE algorithm for musical symbol recognition we, need to determine a suitable centre for each prototypical glyph, and a suitable set of concentric circles and their radii along with the parameters σ_0 and α that control the degree of blurring. We determined these parameters empirically by using a set of 112 systems, containing over 2000 symbols between them, with multiple instances of each symbol as training data. We obtained the systems used for training from the Mutopia Project[4] which is a public music repository, and we manually labelled the symbols to obtain ground-truth data. We then applied the COSFIRE algorithm on these training samples and fine-tuned the parameters to obtain the maximum F-measure for each note.

Once all parameters were set, we applied the COSFIRE filters to other images. Similar to the template matching approach we consider as matches only the local maximum responses that are above a given threshold.

3.2.3 Using hierarchy to improve recognition. In both template matching and COSFIRE approaches, the algorithms indicate a pixel position where a particular symbol is found. While this is sufficient for the localisation of symbols, we note that the symbol recognition improves if we simplify the symbol image by removing from the image any detected symbols by previously applied templates or COSFIRE filters. Thus, we apply connected component analysis to locate all pixels on the same symbol, and remove that symbol from the image.

We further note that the order in which we detect the symbols affects the number of false detections of the symbols. The reason for this is that some symbols have parts which are similar to other symbol parts or may even be contained entirely within other symbols. Thus, a template or a COSFIRE filter which is selective for one symbol may also respond to a more complex or larger symbol, possibly with the same activation such that it is difficult to differentiate between the two. This effect may be observed clearly in the sharp and natural signs, or the quaver and semiquaver rests as illustrated in Figure 4. Therefore, it is sensible that templates and COSFIRE filters are applied in an order that is based on the shape complexity of the concerned symbol, with the more complex symbol being processed first. To determine the ordering, we perform symbol recognition on the training data described above to form a confusion matrix showing the number of correctly and incorrectly classified symbols. Symbols with the larger off-diagonal values are confused more often for other symbols and this is indicative of the order with which symbol recognition should be performed.

Figure 4: Symbols may have parts which are similar to other symbols. Illustrated here are the natural sign and quaver rest (red) superimposed on the sharp sign and the semiquaver rest, respectively.

3.2.4 Using domain knowledge to improve recognition. In music notation, symbols are written following notation rules which allows for standardisation of the notation [20]. Thus, for example, a staccato dot is always placed vertically above or below the note head, while dots that augment the note duration are always on the right hand side of the note and aligned with the note head. This domain knowledge can therefore be used to refine the symbol recognition, adjusting symbol labels such that these match with the expected position of the symbol according to music notation standards.

4 REWRITING THE SCORE IN MUSICXML

After locating and labelling all symbols of interest in the score, we further process the notes and symbols to retrieve the relevant musical information from the score to represent it as a MusicXML file. MusicXML is an XML based digital sheet music interchange and distribution format designed to provide a universal format for Western music notation. The MusicXML file format has similar applications as the MIDI file format, but offers the additional advantage of specifically notating the music, thus capturing information about the stem direction and beams among others as well as allowing for the important distinction between notes and their enharmonic equivalents. The MusicXML file format therefore captures two aspects of the music score, how it should sound and look. The following sections describe our approach of using the fragmented information obtained about the score From the symbol recognition step to represent the score in the MusicXML file format. We refer

[4] http://www.mutopiaproject.org/

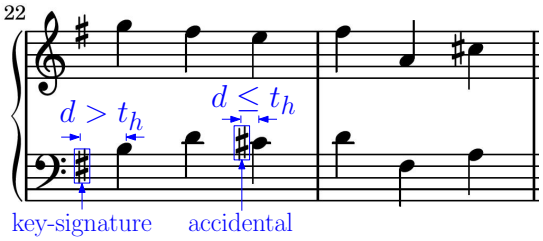


Figure 5: Sharp, flat and natural signs may occur next to notes or as a group, next to the clef. When these signs occur next to notes, they are referred to as accidentals, otherwise, they form a key-signature. A threshold t_h on the horizontal distance from the note-head allows distinction between accidentals and key-signatures.

the reader to the MusicXML documentation[5] for full explanation of this file format.

4.1 Setting the initial score attributes

4.1.1 Key signature. Any sharp or flat sign present in the score alters the note from its natural pitch. Sharp and flat signs can occur at the start of the system where they collectively form a key-signature or on the right-hand side of the note and in-line with the note-head. Although the symbol recognition step locates and identifies all sharps and flats present in each system, we need to distinguish between accidentals and key-signatures. For this reason, we compute the horizontal distance between a note-head and the identified sharp, flat and natural signs and associate with the note any sign located within a threshold t_h from the note as illustrate in Figure 5. We label as a key-signature any remaining sharp or flat signs. Key-signatures always follow the same pattern, such that counting the number of sharp or flat signs in a key-signature is sufficient to determine the key-signature to display in the MusicXML file. Indeed, the MusicXML format encodes the key-signature using the *circle of fifths*, that is, positive numbers are used to represent the number of sharps in the key-signature while negative numbers are used to represent the number of flats.

4.1.2 Time signature. For the scope of this work we assume that the time signature of the score does not change. Thus, the time signature may be located only at the start of the first system of the score and can be identified by a simple OCR algorithm, applied to the local area where the time signature is expected. The MusicXML format breaks the time signature into the number of beats, given by the top numeral of the time signature and the type of beat, given by the bottom numeral of the time signature.

4.1.3 Number of divisions. In musical notation, the duration of a note is expressed in fractions of beats, with the bottom numeral of the time signature defining the beat. A beat may be a simple beat, which we can divide into two equal parts, or a complex beat which we can divide into three equal parts.

[5] http://www.musicxml.com

Table 1: The number of divisions per crotchet note given the shortest note duration in the score

note	minim	crotchet	quaver	semiquaver	demi-semiquaver
r	-1	0	1	2	3

```
<attributes>
    <divisions>8</divisions>
      <key>
        <fifths>-1</fifths>
      </key>
    <time>
      <beats>3</beats>
      <beat-type>4</beat-type>
    </time>
  </attributes>
```

Listing 1: MusicXML attributes for the score in Figure 1

The MusicXML format, rather than using beats, defines the duration of notes in *divisions*, where one division represents the shortest note duration present in the score. In addition to the time signature, the MusicXML format requires the number of divisions that a crotchet note would need, given the shortest note duration in the score. We calculate the number of divisions as:

$$divisions = \begin{cases} 2^r & \text{if simple time} \\ 3 \times 2^r & \text{if compound time} \end{cases} \quad (2)$$

where the exponent r depends on the shortest note duration and is given in Table 1.

The attributes for the score shown in Figure 1, which has a key-signature of one flat, a simple-triple time signature and demi-semiquavers as the shortest note duration are given in Listing 1.

4.2 The note element

The note element of the MusicXML file incorporates within it aspects relating to the sound of the note as well as its appearance. This element, therefore, consists of other elements as required for the particular note. Listing 2 gives the MusicXML representation of the two notes enclosed in the box in Figure 1.

4.2.1 The pitch element. This element describes the sound of the note and consists of three parts, the `step` which states the pitch letter name, the `octave` which describes the octave register of the note, and an optional `alter` which describes the change in pitch from the natural state due to sharp or flat signs.

Thus, we must first obtain the pitch of the notes detected by the symbol recognition step. All notes consist of a note-head and, if applicable, a stem and beams or flags. Since the position of the note-head defines the pitch of the note, we must separate the note-head from the other components of the note. We achieve this by applying binary morphology, opening the note symbol image with a disk element, using the height H_S of the space between the staff lines to determine the size of the disk structuring element. The opening operation allows us to obtain individual note-heads even when the notes appear in contact as happens when the music has

```
<note> <!-- A, crotchet note with a delayed turn -->
  <pitch>
      <step>A</step>
        <octave>4</octave>
    </pitch>
    <duration>8</duration>
    <type>quarter</type>
    <stem>up</stem>
    <notations>
      <ornaments>
          <delayed-turn/>
        </ornaments>
    </notations>
</note>
</note> <!-- the acciaccatura -->
  <grace slash="yes"/>
    <pitch>
      <step>C</step>
        <octave>5</octave>
    </pitch>
    <type>eighth</type>
    <stem>up</stem>
</note>
<note> <!-- B flat, quaver note -->
     <pitch>
      <step>B</step>
        <alter>-1</alter>
        <octave>4</octave>
     </pitch>
     <duration>4</duration>
     <type>eighth</type>
     <stem>up</stem>
</note>
```

Listing 2: MusicXML format of two notes enclosed in the box in Figure 1

chords. We use the vertical distance of the centroid of the note-head from the topmost line of the staff line to determine the position of the note on the staff. In the treble clef, this line represents the pitch F5 while in the bass clef, this line represents the pitch A3. Thus, a clef symbol sets the reference step and octave components of the pitch element for all subsequent note-heads.

Any sharps or flat signs present in the score alter the natural pitch of the note, with a sharp raising the note by one semitone, while a flat lowers the note by one semitone. A third accidental sign, the natural sign, reverts any alternation made to the note by previous sharp or flat signs. In MusicXML format, a sharp sign introduces an `alter` of +1 while a flat sign, introduces an `alter` of −1. The natural sign resets the alter to 0. In music notation, if the sharp or flat sign is part of a key-signature, then all notes in the score that have the same pitch letter name, irrespective of the octave register are affected by the sign. On the other hand, accidentals only affect notes which are in the same bar and which have the same pitch and octave-register as the note to which the accidental is applied. Moreover, within the bar, the accidental has priority over the alterations introduced by the key-signature. Thus, in writing the MusicXML file, we first adjust the note alters for all notes affected by the key-signature, following which, we apply any additional changes due to accidentals. Line 27 in Listing 2 shows the `alter` required to notate the note B as a flattened note as per the key-signature.

Table 2: The relative duration of a note as a fraction of a crotchet.

note	semibreve	minim	crotchet	quaver	semiquaver
b	4	2	1	$1/2$	$1/4$

4.2.2 The note duration. The MusicXML format defines the duration of each note as a fraction of the divisions per crotchet note as defined in the score attributes. Thus, the duration of each note is $b_i \times divisions$ where b_i is the duration of the note relative to the crotchet note and whose values are given in Table 2.

The duration of the note may increase with the addition of dots on the right-hand side of the note-head. If a note has dots within a horizontal distance of t_h, the duration of the notes is increased by a factor of $(2^{n+1}-1)/2^n$ where n is the number of dots associated with the note.

4.2.3 The stem element. This element specifies the direction of the stem. Since notes obtained from the score are already typeset, we retain the stem direction used in the score, distinguishing the stem direction by enclosing the note within a bounding box and noting the position of the note-head within the bounding box.

4.2.4 The notation element. In MusicXML, the notation element describes any articulations or ornaments that may apply to the note. In this work, we focus on staccato and staccatissimo articulation symbols, the pause symbol as well as the trill, turn and mordent ornaments. These symbols occur either above or below the note-head. Thus, to determine whether a note has any such symbol acting upon it, we place a window around each note, where the width of the window is the width of the note-head, and the height is set empirically to twice the staff height. We express any symbol found within this window using the appropriate MusicXML syntax.

In the case of delayed turns such as the first turn ornament in Figure 1, the turn ornament is placed between two notes rather than directly above the note. In such cases, the ornament is not captured in the vertical window. Thus, for any turn symbol detected by the symbol recognition step and not associated with a note, we locate the two notes nearest to the turn symbol. If they are within an acceptable distance, we associate the turn with the left-most note of the pair, labelling the ornament as a delayed-turn as illustrated in Lines 10-12 in Listing 2.

4.3 Grace notes

Grace notes such as the acciaccatura and the appogiatura differ from other ornaments since their symbols are similar to the note notation, that is, they are pitched. Their representation in the MusicXML format is therefore similar to other notes. However, since the duration of these grace notes depends on their interpretation, the duration of the grace note is not specified in the MusicXML note attributes as illustrated in Lines 15-23 of Listing 2.

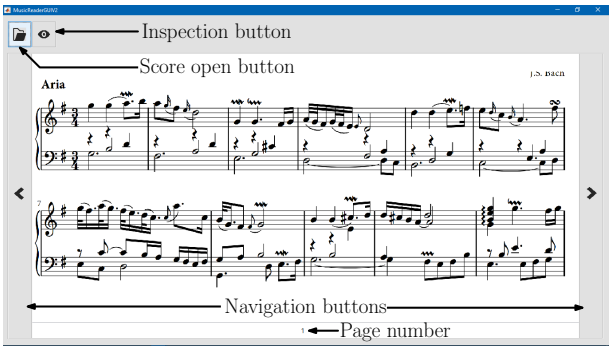


Figure 6: The main interface used to allow users to inspect ornaments.

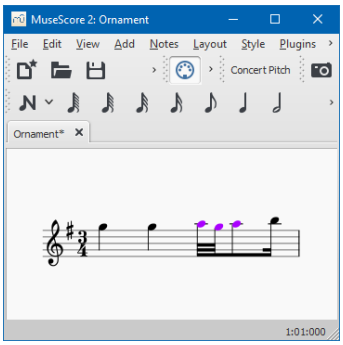


Figure 7: Inspecting the ornament in the first bar.

5 EXPRESSING ORNAMENTS IN FULL

Through writing the MusicXML file, we associate each ornament in the score with the note on which it acts. Moreover, when writing the MusicXML file, we keep track of the line numbers associated with each note in the score. This allows us to create a copy of the MusicXML file in which we replace the lines defining an ornamented note with other lines containing notes definitions for the expressed ornament.

In case of grace notes, this requires the removal of the `grace` element and the insertion of the `duration` element which gives the grace note its duration. Since the duration of all notes within the bar must remain the same, we also need to reduce the duration of the note after the grace note by the same amount. We use musical theory to determine the duration of the grace note and the harmony note. This depends on whether the grace note is an appoggiatura or an acciccatura, whether the harmony note is a simple note or one which is dotted or tied to subsequent notes [20].

Mordent, turn, and trill ornaments require more complex modifications, inserting two or more notes to the MusicXML file as applicable and according to music theory. For example, the mordent is expressed as three notes, performing a rapid alternation between the indicated note and the note above it (upper mordent) or the one below it (lower mordent) [20]. Thus, we insert two copies of the `note` element, the second of which we adjust the pitch by increasing or decreasing the `step` accordingly. We also adjust the duration of all three notes such that the overall duration of the expressed mordent remains the same as the original note duration while obtaining the rapid alternation required at the first two notes. If the original note is not a dotted note, the first two notes of the expressed ornament are each assigned $1/8$ of the total duration, with the third note getting the remaining $3/4$ of the total duration. If however, the original note is a dotted note, then the ratios are adjusted to $1/6$ and $2/3$, respectively [20]. A similar approach is adopted for turns and trills.

5.1 User interface

While we may express all ornaments in the score without needing user interaction, we opt for a user-interface to give the music learner the possibility to select the ornaments that are required to be expressed in full. Such an interface gives the learner the possibility of switching between the fully expressed ornaments and their symbolic representation until the music learner gains confidence in the execution of the ornaments. Thus, as shown in Figure 6, scores are loaded onto the system through the key-press of the load button. If the score selected is a new score, then the score pre-processing, and symbol recognition steps are executed, writing the score as a MusicXML file prior to displaying the score image. This will create an intermediary array in which the note (x, y) image co-ordinates are mapped to the corresponding lines in the MusicXML file. If the score has been previously processed, the image pre-processing and symbol recognition steps are not performed again.

As shown in Figure 6 the score is presented to the user using a two-system display which we found best for readability of the score [4]. The user can navigate through the entire score using the navigation buttons, or jump to particular pages by typing in the page number. A key-press on the inspection icon will change the cursor to a cross-hair pointer, prompting the user to select an ornamented note by clicking on it. Upon clicking on an ornament, the interface extracts the (x, y) coordinates selected by the user and compares these to the ornament locations detected through the symbol recognition step. The closest ornament to the point clicked by the user is established as the selected ornament. Through the preparation of the MusicXML files, this ornament is associated with a note, the bar number in which it occurs, the staff line on which it is written as well as the MusicXML lines corresponding to the note. From the MusicXML file we extract the lines corresponding to all notes in the same bar and from the same staff line as the selected note. These are re-written in a new MusicXML file which we name Ornament.xml and in which we replace the ornament symbol with the expressed ornament. This new file is automatically imported into a music reader such as MuseScore, thereby allowing the user to view the expressed ornament as shown in Figure 7. We highlight the notes pertaining to the expressed ornament by setting the colour of the note-heads to purple, making it easier for the user to identify the inserted notation.

Since some ornaments may have more than one interpretation, the user may at this point, make adjustments to the notations using the music reader interface. If adjustments are made, then the score snippet should be exported as a MusicXML file. The user may then

opt to retain the score with the ornament symbol or update the MusicXML file with the expressed ornament. If the user chooses the latter option, the note elements in the Ornament.xml file replace those in the original file.

Once the user finishes exploring the score and its ornaments, the MusicXML file may be exported to any music reader or exported as a pdf document for printing.

6 EVALUATION METHODOLOGY

The performance of the ornament expression rests on the ability of the symbol recognition algorithms in localising and correctly labelling the symbols in the score. We evaluate the performance of the symbol recognition step by applying the COSFIRE filter symbol recognition approach to a labelled data set consisting of 124 staves with 5000 individually labelled symbols, with at least ten instances of each symbol. We quantify the results obtained by the symbol recognition algorithm with the ground truth symbols, by counting the number of true matches, missed symbols and false detections in each case, from which we obtain measures of the precision, recall and F-score. To determine the effect of the inclusion of symbol hierarchy and domain knowledge, we quantify the performance of the algorithm after performing the symbol recognition following a specific hierarchy and again, when domain knowledge is introduced. For comparison purposes, the symbol recognition is also performed using template matching [21]. For fairness of evaluation, hierarchical ordering and domain knowledge are also introduced to the template matching approach. In this evaluation, the 124 labelled staves were typeset using two different fonts, namely, the Emmentaler and the Bravura[6] font sets. Note that the COSFIRE and the template matching algorithms were trained on glyphs obtained from Emmentaler font set alone and thus, evaluation on the Bravura font set give a measure of the adaptability of the algorithms to changes in symbol fonts.

The next step in the evaluation of the ornament expression is that of determining whether the symbolic information extracted from the score can be successfully represented in the MusicXML file format. Thus, we manually adjust for any incorrect symbols from the symbol recognition step and compare the MusicXML files with the original score, noting discrepancies in the notation.

We then perform symbol recognition and ornament expression on a score containing different ornaments to verify that the expressed ornaments follow a reasonable and musical interpretation. To evaluate this, we select pieces from reputable sources such as publications from the London College of Music[7] and the Associated Board of the Royal Schools of Music[8] which have annotated ornaments and compare our algorithmic interpretation with the annotated interpretation. Lastly, we demonstrate the user interface to five music students as well as a music teacher with over 50 years experience, to gauge their response to the ornament expression tool proposed in this work.

[6]https://www.smufl.org/
[7]http://lcme.uwl.ac.uk/home
[8]https://mt.abrsm.org/en/home

7 RESULTS

The results obtained for the symbol recognition step are summarised in Table 3. We observe that the two symbol recognition approaches perform better with the Emmentaler font set than with the Bravura font set. Such a result was expected since we used the Emmentaler font set to create the templates and prototypes required by the two algorithms.

If we consider the first, un-ranked approach evaluated, we note that the COSFIRE filter outperforms the template matching approach. Here, the errors of the COSFIRE filter approach are concentrated around two sets of symbols, namely the quaver and semiquaver symbols and the staccato and staccatissimo symbols. In the case of the quaver-semiquaver pairs, the COSFIRE incorrectly labelled 22% of the semiquavers as quavers. This high misclassification is most likely occurring because the semiquaver symbol contains the quaver symbol and is, therefore, a match to the quaver prototype. Indeed, the template matching approach also has a similar poor performance at quaver-semiquaver pairs. The template matching approach has a better performance at the staccato and staccatissimo symbols, although the performance is significantly lower for natural, flat, sharp and acciaccatura symbols.

Performing the symbol recognition using the hierarchical ranking of the symbols, improves the performance of the symbol recognition for both approaches. The improvement is mainly due to an improvement in the quaver-semiquaver symbols since these are now all correctly classified.

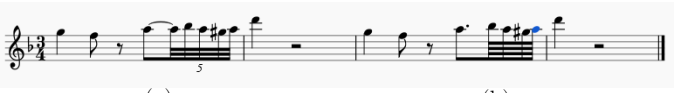
Introducing domain knowledge helps to improve the results further since the expected position of the symbol is used to determine whether the symbol is correctly labelled. The source of error in the COSFIRE filter approach remains the staccato and staccatissimo symbols, although the number of correct classifications of the staccato symbol improves from 4% in the initial evaluation to 54% with the introduction of hierarchical ranking and domain knowledge. With a classification rate of 94%, the template matching performs slightly better than the COSFIRE filter approach for this symbol. However, template matching lags behind with the classification of the acciaccatura, natural and flat signs, with true classification rates of 60.42%, 14.54% and 24.31% respectively in comparison to the COSFIRE classification rates of 100%, 84.32% and 84.31%.

The ornaments expressed algorithmically conformed in pitch with the ornament interpretation guidelines. However, there were some discrepancies in the note durations set by the algorithm in comparison to those in the guidelines. For example, the ABRSM suggests that the delayed turn marked (f) in Figure 1 should be expressed as shown in Figure 8(a). Our interpretation, however, is at a faster pace as shown in Figure 8(b) which conforms with the Schirmer interpretation in Figure 1. Similar rhythmic differences were observed in mordents an appoggiaturas. Thus, although different, the interpretations were not incorrect.

The user interface and the concept of representing ornaments in full at a learning stage was well received. One student in particular commented that seeing the ornament in full would help her understand how notes would fit in together.

Table 3: Comparison of the Precision and Recall values (in percentages) obtained by the Template Matching and COSFIRE algorithm.

	Template Matching						COSFIRE filters					
	Emmentaler			Bravura			Emmentaler			Bravura		
	Precision	Recall	**F-score**	Precision	Recall	**F-score**	Precision	Recall	**F-score**	Precision	Recall	**F-score**
Without Ranking	77.23	96.06	**85.63**	79.21	93.12	**85.61**	87.94	98.44	**92.89**	79.03	96.87	**87.05**
With Ranking	84.03	95.77	**89.37**	89.05	92.74	**90.86**	90.54	98.34	**94.28**	85.34	96.58	**90.51**
Domain knowledge	86.70	95.77	**91.14**	91.87	92.74	**92.30**	95.63	98.34	**96.97**	87.34	96.58	**91.74**



(a) (b)

Figure 8: Interpretations of a turn (a) as suggested by the ABRSM and (b) our interpretation.

8 SUMMARY AND CONCLUSIONS

In this paper we present an optical music recognition system which takes as input musical sheet music containing ornament symbols and generates a MusicXML file representing the score. We also present a user interface through which the user may select ornamented notes from the score image and view an expressed version of the ornamented note. Moreover, the interface allows the user to expand the score such that the ornamented notes are written in full.

We evaluate the various steps of the OMR system, starting with the symbol recognition algorithm where we compare two approaches namely a template matching and a COSFIRE filter approach. The results obtained show that hierarchical ordering the symbols as well as domain knowledge helps to reduce the number of false detections and hence improves the detection rate in both approaches. With more complex symbols the COSFIRE outperformed the template matching approach. However, with the smaller and simpler symbols such as the staccato and staccatissimo symbols, the template matching approach offered better results. The reason for this may be because the symbols are relatively small and do not contain any particular ink stroke pattern which the COSFIRE prototype can model. Thus, the use of the COSFIRE filter is less attractive than template matching for such symbols. More complex symbols such as the accidentals, however, have complex patterns which are more effectively captured with the COSFIRE approach than with the template matching. This suggests the need of a two-tier symbol recognition, with the simpler, template based approach being used for simple symbols, using the COSFIRE filter for more complex symbols.

In this work, the MusicXML file is written in full through our algorithms and so, the format of the file does not change. This allows us to index the file by using line numbers. The system may be made more flexible if it allows the user to import pre-existing MusicXML files, bypassing the music recognition process. To allow for such adaptation, the search and replacement of ornaments requires full use of XML functionality and in future, we will look into the use of XML query techniques to allow for differences in file formatting. The expression of the ornaments in full provides for at least one plausible interpretation of the ornament. In future, this may be improved by allowing the user to view and select from a number of different interpretations. We also plan on increasing the data set of symbols to make the conversion of image scores into the MusicXML file format more robust to a wider range of music scores. This would allow us to evaluate the proposed ornament expression tool "in the wild" by giving it to students to better gauge its effect on learning over a longer period of time.

REFERENCES

[1] G. Azzopardi and N. Petkov. 2013. Trainable COSFIRE Filters for Keypoint Detection and Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 2 (Feb. 2013), 490–503.

[2] David Bainbridge and Tim Bell. 2001. The Challenge of Optical Music Recognition. *Computers and the Humanities* 35, 2 (01 May 2001), 95–121.

[3] A. Baró, P. Riba, J. Calvo-Zaragoza, and A. Fornés. 2017. Optical Music Recognition by Recurrent Neural Networks. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 02. 25–26.

[4] Alexandra Bonnici, Stefania Cristina, and Kenneth P Camilleri. 2017. Preparation of Music Scores to Enable Hands-free Page Turning Based on Eye-gaze Tracking. In *Proceedings of the 2017 ACM Symposium on Document Engineering*. ACM, 201–210.

[5] Artur Capela, Ana Rebelo, Jaime S Cardoso, and Carlos Guedes. 2008. Staff Line Detection and Removal with Stable Paths.. In *SIGMAP*. 263–270.

[6] G. Chen, L. Zhang, W. Zhang, and Q. Wang. 2010. Detecting the Staff-Lines of Musical Score with Hough Transform and Mathematical Morphology. In *Proc. Int. Conf. Multimedia Technology*. 1–4.

[7] Liang Chen, Rong Jin, and Christopher Raphael. 2015. Renotation from optical music recognition. In *Mathematics and Computation in Music*. Springer, 16–26.

[8] Bertrand Coüasnon and Jean Camillerapp. 1994. Using grammars to segment and recognize music scores. In *International Association for Pattern Recognition Workshop on Document Analysis Systems*. 15–27.

[9] Ichiro Fujinaga. 2004. Staff detection and removal. In *Visual Perception of Music Notation: On-Line and Off Line Recognition*. IGI Global, 1–39.

[10] Basilios Gatos, Ioannis Pratikakis, and Stavros J Perantonis. 2004. An adaptive binarization technique for low quality historical documents. In *International Workshop on Document Analysis Systems*. Springer, 102–113.

[11] Jan Hajic jr and Matthias Dorfer. 2017. Prototyping full-pipeline optical music recognition with musicimarker. *The 18th International Society for Music Information Retrieval Conference* (2017).

[12] GueeSang Lee et al. 2015. Music Score Recognition Based on a Collaborative Model. *International Journal of Multimedia and Ubiquitous Engineering* 10, 8 (2015), 379–390.

[13] Bharath R Modayur, Visvanathan Ramesh, Robert M Haralick, and Linda G Shapiro. 1993. MUSER: A prototype musical score recognition system using mathematical morphology. *Machine Vision and Applications* 6, 2-3 (1993), 140–150.

[14] Tam Nguyen and Gueesang Lee. 2015. A Lightweight and Effective Music Score Recognition on Mobile Phones. *JIPS* 11, 3 (2015), 438–449.

[15] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre RS Marcal, Carlos Guedes, and Jaime S Cardoso. 2012. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval* 1, 3 (2012), 173–190.

[16] F. Rossant and I. Bloch. 2005. Optical music recognition based on a fuzzy modeling of symbol classes and music writing rules. In *Proc. IEEE Int. Conf. Image Processing 2005*, Vol. 2. II–538–41.

[17] Melanie Spanswick. 2014. A few thoughts on ornaments. (2014). https://melaniespanswick.com/2014/11/17/a-few-thoughts-on-ornaments/

[18] B. Su, S. Lu, U. Pal, and C. L. Tan. 2012. An Effective Staff Detection and Removal Technique for Musical Documents. In *Proc. 10th IAPR Int. Workshop Document Analysis Systems*. 160–164.

[19] T. Tambouratzis. 2011. Identification of key music symbols for optical music recognition and on-screen presentation. In *Proc. Int. Joint Conf. Neural Networks*. 1935–1942.

[20] Eric Robert Taylor. 1992. *The AB guide to music theory Part I.* Associated Board of the Royal School of Music.

[21] Fubito Toyama, Kenji Shoji, and Juichi Miyamichi. 2006. Symbol recognition of printed piano scores with touching symbols. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, Vol. 2. IEEE, 480–483.

[22] Eelco van der Wel and Karen Ullrich. 2017. Optical Music Recognition with Convolutional Sequence-to-Sequence Models. In *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*. 731–737.

[23] Muriel Visaniy, Van Cuong Kieu, Alicia Fornés, and Nicholas Journet. 2013. ICDAR 2013 music scores competition: Staff removal. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 1407–1411.